

Salary Range Prediction Model

Project-6



July 12, 2024

Spinnaker analytics

**Table of Contents**

1. **Project Overview**
2. **Dataset Description**
3. **Exploratory Data Analysis (EDA)**
   * Distribution of Salary Ranges
   * Average Salary by Job Category
   * Predicted vs. Actual Salary Ranges
   * Residual Plot
   * Distribution of Prediction Errors
4. **Model Building**
   * Feature Selection
   * Model Performance
5. **Model Evaluation**
   * Performance Metrics
6. **Hyperparameter Tuning**
   * Best Parameters
   * Validation Set Error
7. **Conclusion**
8. **Source code**

**1. Project Overview**

**Purpose**

The purpose of this project is to develop a predictive model for estimating salary ranges for job listings. The model aims to enhance recruitment processes, improve candidate experience, and support strategic decision-making in talent management.

**Objectives**

* **Enhanced Recruitment Process:** Align job postings with market standards to attract top-tier talent and optimize hiring costs.
* **Improved Candidate Experience:** Provide transparent salary expectations to enhance candidate engagement and satisfaction, leading to higher acceptance rates.
* **Strategic Decision-Making:** Empower HR professionals and organizational leaders with reliable salary forecasts for informed budget allocation and resource planning.

**2. Dataset Description**

**Overview**

* **Source:** NYC job postings dataset (Jobs\_NYC\_Postings.xlsx).
* **Size:** 5120 entries with 30 columns.
* **Data Types:** Integers, floats, objects, and datetime64.

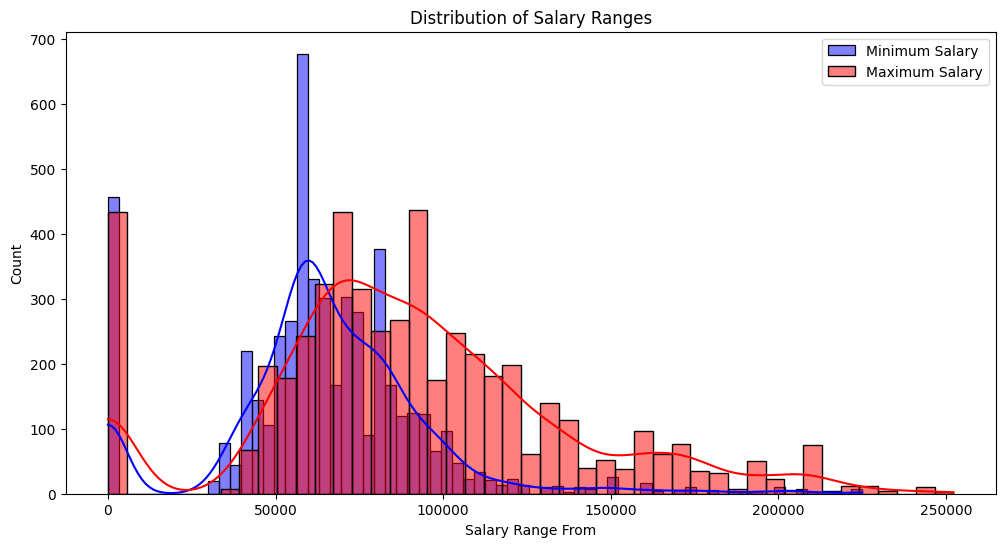
**Preprocessing**

* **Missing Values:** Handled appropriately by dropping rows with missing salary information and converting salary columns to numeric types.
* **Dataset Statistics:** Basic statistics and data types overview provided.

**3. Exploratory Data Analysis (EDA)**

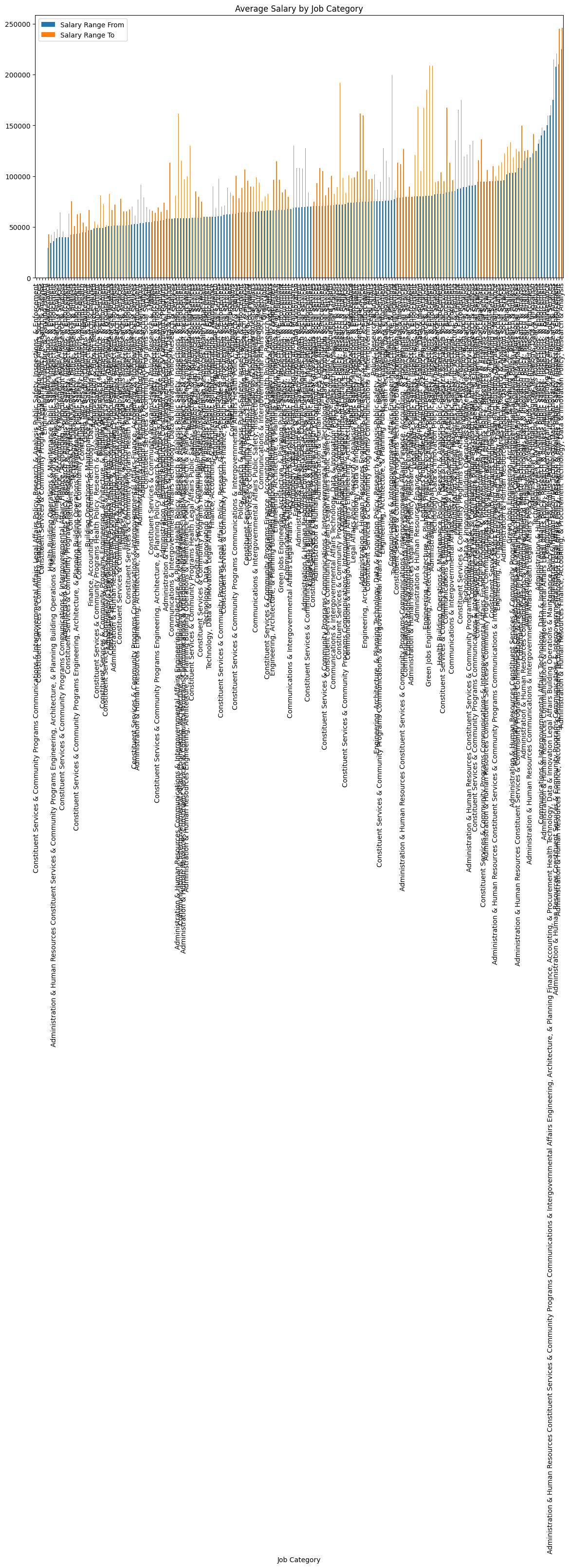
**Distribution of Salary Ranges**

* **Insight:** The distribution shows peaks at certain salary ranges, indicating common trends in job postings.



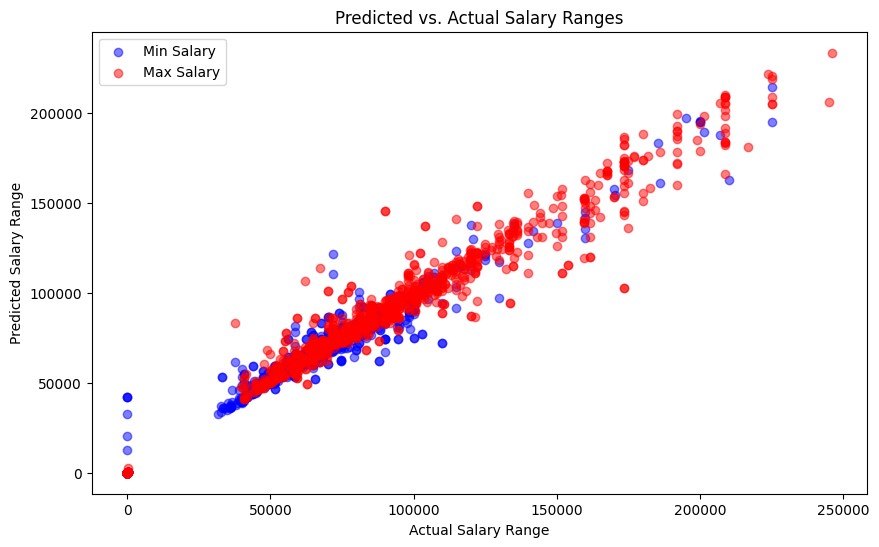
**Average Salary by Job Category**

* **Analysis:** Different job categories exhibit varying average salary ranges, highlighting disparities across sectors.



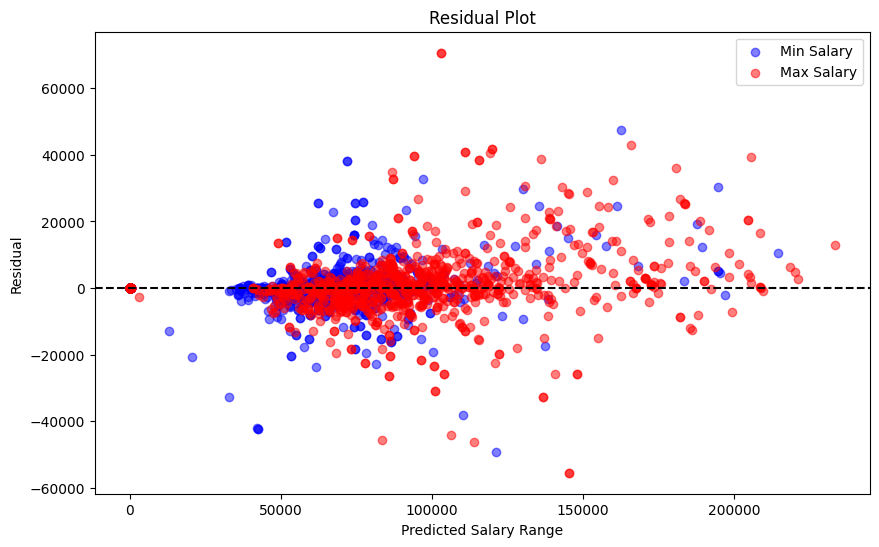
**Predicted vs. Actual Salary Ranges**

* **Visualization:** Scatter plot comparing predicted vs. actual salary ranges, demonstrating model accuracy.



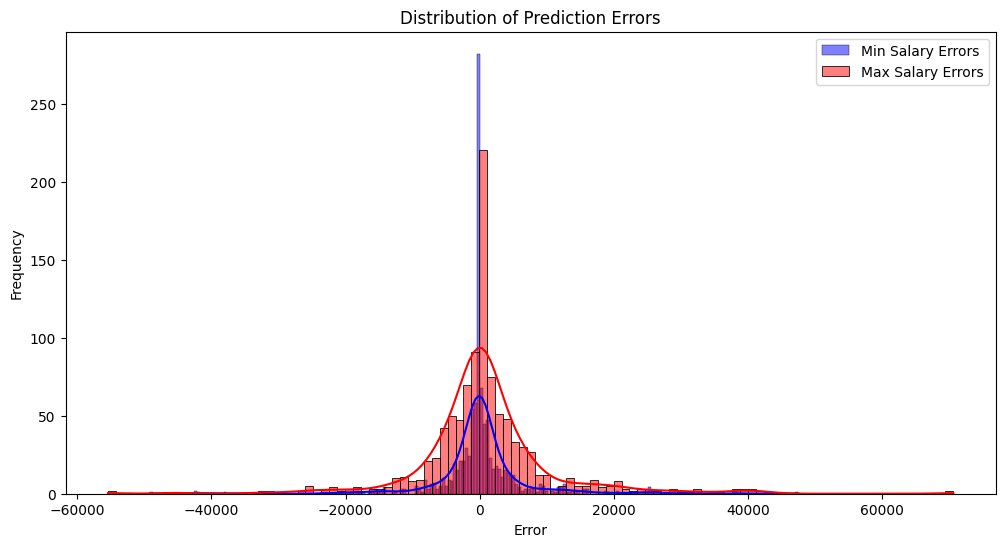
**Residual Plot**

* **Analysis:** Residuals are symmetrically distributed around zero, indicating balanced predictions with minimal bias.



**Distribution of Prediction Errors**

* **Insight:** Errors are distributed with a slight skew, highlighting areas for potential model improvement.



**4. Model Building**

**Feature Selection**

* **Selected Features:** Job-related details, requirements, and other relevant attributes considered for model training.
* **Data Transformation:** Categorical features converted into numerical representations using one-hot encoding.

**Model Performance**

* **Mean Absolute Error (Min Salary):** 3395.31
* **Mean Absolute Error (Max Salary):** 5831.20
* **Analysis:** The model demonstrates moderate accuracy in predicting salary ranges based on selected features.

**5. Model Evaluation**

**Performance Metrics**

* **Mean Squared Error (Min Salary):** 47816441.97
* **Root Mean Squared Error (Min Salary):** 6914.94
* **R-squared (Min Salary):** 0.95
* **Analysis:** The model explains 95% of the variance in minimum salary predictions, indicating strong predictive capability.

**6. Hyperparameter Tuning**

**Best Parameters**

* **Parameters:** {'max\_depth': 40, 'max\_features': 'log2', 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 196}
* **Mean Absolute Error (Tuned Model):** 322.77

**Validation Set Error**

* **Validation Error:** 271.34
* **Conclusion:** Tuned parameters significantly improve model accuracy, enhancing predictive performance.

**7. Conclusion**

**Summary**

The salary range prediction model effectively estimates salary ranges based on job attributes and requirements, supporting improved recruitment strategies and decision-making processes in talent management.

**Future Steps**

* **Feature Refinement:** Further refinement of features to enhance model accuracy.
* **Hyperparameter Optimization:** Continued tuning of model parameters for optimal performance.
* **Integration:** Integration of the model into recruitment processes for real-time salary estimation.

**Source Codes:**

Step 1: Load and Inspect the Dataset

import pandas as pd

# Load the dataset

file\_path = 'Jobs\_NYC\_Postings.xlsx'

df = pd.read\_excel(file\_path)

# Display basic information about the dataset

print(df.info())

print(df.head())

Step 2: Data Preprocessing

# Handle missing values

df = df.dropna(subset=['Salary Range From', 'Salary Range To'])  # Remove rows with missing salary info

# Convert salary columns to numeric if not already

df['Salary Range From'] = pd.to\_numeric(df['Salary Range From'], errors='coerce')

df['Salary Range To'] = pd.to\_numeric(df['Salary Range To'], errors='coerce')

# Fill remaining missing values if needed

df = df.fillna('Unknown')

# Display basic statistics

print(df.describe())

Step 3: Exploratory Data Analysis (EDA)

import matplotlib.pyplot as plt

import seaborn as sns

# Distribution of salary ranges

plt.figure(figsize=(12, 6))

sns.histplot(df['Salary Range From'], kde=True, color='blue', label='Minimum Salary')

sns.histplot(df['Salary Range To'], kde=True, color='red', label='Maximum Salary')

plt.legend()

plt.title('Distribution of Salary Ranges')

plt.show()

# Job categories and their average salaries

plt.figure(figsize=(14, 7))

avg\_salaries = df.groupby('Job Category')[['Salary Range From', 'Salary Range To']].mean().sort\_values(by='Salary Range From')

avg\_salaries.plot(kind='bar', figsize=(14, 7))

plt.title('Average Salary by Job Category')

plt.show()

plt.figure(figsize=(10, 6))

plt.scatter(y\_test\_min, y\_pred\_min, color='blue', label='Min Salary', alpha=0.5)

plt.scatter(y\_test\_max, y\_pred\_max, color='red', label='Max Salary', alpha=0.5)

plt.xlabel('Actual Salary Range')

plt.ylabel('Predicted Salary Range')

plt.title('Predicted vs. Actual Salary Ranges')

plt.legend()

plt.show()

plt.figure(figsize=(10, 6))

plt.scatter(y\_pred\_min, y\_test\_min - y\_pred\_min, color='blue', label='Min Salary', alpha=0.5)

plt.scatter(y\_pred\_max, y\_test\_max - y\_pred\_max, color='red', label='Max Salary', alpha=0.5)

plt.axhline(y=0, color='black', linestyle='--')

plt.xlabel('Predicted Salary Range')

plt.ylabel('Residual')

plt.title('Residual Plot')

plt.legend()

plt.show()

errors\_min = y\_test\_min - y\_pred\_min

errors\_max = y\_test\_max - y\_pred\_max

plt.figure(figsize=(12, 6))

sns.histplot(errors\_min, kde=True, color='blue', label='Min Salary Errors')

sns.histplot(errors\_max, kde=True, color='red', label='Max Salary Errors')

plt.xlabel('Error')

plt.ylabel('Frequency')

plt.title('Distribution of Prediction Errors')

plt.legend()

plt.show()

Step 4: Model Building

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

# Select relevant columns and drop rows with missing salary information

df = df[['Job ID', 'Agency', 'Business Title', 'Civil Service Title', 'Title Classification', 'Job Category',

         'Full-Time/Part-Time indicator', 'Career Level', 'Salary Range From', 'Salary Range To', 'Salary Frequency',

         'Work Location', 'Division/Work Unit', 'Minimum Qual Requirements', 'Preferred Skills']]

df = df.dropna(subset=['Salary Range From', 'Salary Range To'])

# Convert categorical columns to numerical

categorical\_cols = ['Agency', 'Business Title', 'Civil Service Title', 'Title Classification', 'Job Category',

                    'Full-Time/Part-Time indicator', 'Career Level', 'Salary Frequency', 'Work Location',

                    'Division/Work Unit', 'Minimum Qual Requirements', 'Preferred Skills']

df = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

# Define features and target

X = df.drop(columns=['Salary Range From', 'Salary Range To'])

y\_min = df['Salary Range From']

y\_max = df['Salary Range To']

# Split the data into training and testing sets

X\_train, X\_test, y\_train\_min, y\_test\_min, y\_train\_max, y\_test\_max = train\_test\_split(X, y\_min, y\_max, test\_size=0.2, random\_state=42)

# Initialize the model

model\_min = RandomForestRegressor(n\_estimators=100, random\_state=42)

model\_max = RandomForestRegressor(n\_estimators=100, random\_state=42)

# Train the model

model\_min.fit(X\_train, y\_train\_min)

model\_max.fit(X\_train, y\_train\_max)

# Predict on the test set

y\_pred\_min = model\_min.predict(X\_test)

y\_pred\_max = model\_max.predict(X\_test)

# Evaluate the model

mae\_min = mean\_absolute\_error(y\_test\_min, y\_pred\_min)

mae\_max = mean\_absolute\_error(y\_test\_max, y\_pred\_max)

(mae\_min, mae\_max)

Step 5: Model Evaluation

print(f'Mean Absolute Error (Min Salary): {mae\_min:.2f}')

print(f'Mean Absolute Error (Max Salary): {mae\_max:.2f}')

import pandas as pd

# Load your Excel file

file\_path = 'Jobs\_NYC\_Postings.xlsx'  # Replace with your actual file path

df = pd.read\_excel(file\_path)

# Display the column names to identify the date-related column

print(df.columns)

import pandas as pd

# Load your Excel file

file\_path = 'Jobs\_NYC\_Postings.xlsx'  # Replace with your actual file path

df = pd.read\_excel(file\_path)

# Display the column names to identify the date-related column

print(df.columns)

Hyperparameter tuning

import numpy as np

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_absolute\_error

from scipy.stats import randint

# Assuming df is DataFrame with relevant features and target

# Example of adding Log\_Salary\_From and Log\_Salary\_To features

df['Log\_Salary\_From'] = np.log(df['Salary Range From'] + 1)  # Adding 1 to avoid log(0) and handling negative values

df['Log\_Salary\_To'] = np.log(df['Salary Range To'] + 1)      # Adding 1 to avoid log(0) and handling negative values

# Verify column names and existence

required\_columns = ['# Of Positions', 'Log\_Salary\_From', 'Log\_Salary\_To']  # Ensure all required columns are present

for col in required\_columns:

    if col not in df.columns:

        raise KeyError(f"Column '{col}' not found in DataFrame.")

# Define features and target

X = df[required\_columns]

y = df['Salary Range To']

# Check for infinite or large values

if np.any(np.isinf(X)) or np.any(np.abs(X) > 1e10):  # Adjust threshold as per data

    raise ValueError("Input data contains infinite or very large values.")

# Split data into training and validation sets

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the model

rf = RandomForestRegressor(random\_state=42)

# Define parameter grid for RandomizedSearchCV

param\_dist = {

    'n\_estimators': randint(50, 200),

    'max\_depth': [None, 10, 20, 30, 40, 50],

    'min\_samples\_split': randint(2, 20),

    'min\_samples\_leaf': randint(1, 20),

    'max\_features': ['sqrt', 'log2']  # Adjusted max\_features options

}

# Perform RandomizedSearchCV

random\_search = RandomizedSearchCV(estimator=rf, param\_distributions=param\_dist, n\_iter=100,

                                   cv=5, scoring='neg\_mean\_absolute\_error', random\_state=42, n\_jobs=-1, error\_score=np.nan)

random\_search.fit(X\_train, y\_train)

# Print best parameters and best score

print("Best Parameters:", random\_search.best\_params\_)

print("Best Mean Absolute Error:", -random\_search.best\_score\_)

# Evaluate best model on validation set

best\_model = random\_search.best\_estimator\_

y\_pred = best\_model.predict(X\_val)

mae = mean\_absolute\_error(y\_val, y\_pred)

print("Mean Absolute Error on Validation Set:", mae)

from sklearn.metrics import mean\_squared\_error, r2\_score

mse\_min = mean\_squared\_error(y\_test\_min, y\_pred\_min)

rmse\_min = mean\_squared\_error(y\_test\_min, y\_pred\_min, squared=False)

r2\_min = r2\_score(y\_test\_min, y\_pred\_min)

print(f'Mean Squared Error (Min Salary): {mse\_min:.2f}')

print(f'Root Mean Squared Error (Min Salary): {rmse\_min:.2f}')

print(f'R-squared (Min Salary): {r2\_min:.2f}')